Homework 1

Group 14 Contributions: Henrique Machado (ist103202) completed Question 1, Jaime Silva (ist112464) completed Question 2, and Gil Arroteia (ist112466) completed Question 3.

Question 1

1(a)

The test accuracy of the model with the best performance on the validation set is 61.1% it's validation `results/single_perceptron/Q1-perceptron-save.pkl` `results/single_perceptron/Q1-perceptron-accs.pdf` For this exercise we implemented the update_weight, train_epoch, predict, and evaluate methods of the Perceptron class in `skeleton_code/hw1-perceptron.py` and run that file.

2(a)

The test accuracy of the model with the best performance on the validation set is 69.6% it's validation `results/logistic_regression/Q1-2-logistic-a.pkl` `results/logistic_regression/Q1-2-logistic-a.pdf` exercise we created a logistic regression class in `skeleton_code/hw1-logistic_regression.py` and run it with `experiments/logistic_regression_run.py -q a` .

2(b)

We decided to go with max pooling with a 2×2 window: Initially, we considered using 4×4 Average Pooling to reduce the feature space. However, experiments showed that while training was significantly faster, accuracy dropped because the images became "washed out" (e.g., a distinct white pixel averaged with black background becomes a faint gray value, losing contrast).

To solve this, we implemented 2×2 Max Pooling. Instead of averaging, this method selects the maximum value in each 2×2 block. This ensures that if a stroke (white pixel) exists in a block, the signal is preserved rather than diluted. This approach improved robustness to small shifts in letter positioning and reduced the input dimensionality from 784 to 196. Empirically, this reduced training time from ~1m 20s to ~35s with negligible loss in accuracy.

To test this, we can add the `--dataset-format pooled` flag in `experiments/logistic_regression_run.py` to use the pooled dataset.

2(c)

Dataset Structure	Learning Rate	L2 Penalty	Best Validation Accuracy
original	0.0001	0.00001	0.6940
original	0.0001	0.0001	0.6951
original	0.001	0.00001	0.7192
original	0.001	0.0001	0.7195
original	0.01	0.00001	0.6992

original	0.001	0.0001	0.7195
original	0.01	0.00001	0.6992
original	0.01	0.0001	0.6948
pooled	0.0001	0.00001	0.6668
pooled	0.0001	0.0001	0.6680
pooled	0.001	0.00001	0.7091
pooled	0.001	0.0001	0.7095
pooled	0.01	0.00001	0.7086
pooled	0.01	0.0001	0.7067

The test accuracy of the configuration with the best validation accuracy was 72.1%.

3(a)

The test accuracy of the configuration with the best validation accuracy was 87.8% and the validation accuracy was 87.8% at epoch 20 as shown in `results/multilayer_perceptron/Q1-3-scores.json`. That model was saved in `results/multilayer_perceptron/Q1-3-mlp.pkl`. The train and validation accuracy curves are `results/multilayer_perceptron/Q1-3-mlp-loss.pdf`

Question 2

This document consolidates the key findings from all Question 2 experiments (PyTorch FFNs on EMNIST Letters) so we can quickly craft the final report answers.

Implementation Warmup

- `FeedforwardNetwork` in `skeleton_code/hw1-ffn.py` now supports arbitrary depth, configurable activation (`relu`/`tanh`), dropout, and weight decay.
- Training utilities (`train_batch`, `evaluate`, CLI options) were generalized to save metrics/plots and to run scripted sweeps without modification.
- Sanity runs with width = 16 confirmed the pipeline before large sweeps.

Part 2 – Width Study (Toward Infinite Width)

Grid Configuration

- Hidden widths: 16, 32, 64, 128, 256.
- Learning rates: {1e-4, 3e-4, 1e-3, 3e-3}.
- Dropout: {0, 0.2}; Weight decay: {0, 1e-4}.
- Optimizer: Adam; activation: ReLU; epochs: 30; batch size: 64.
- Automation script: `experiments/run_width_grid.py` (outputs under `figures/width_study/`).

Best Validation Accuracy Per Width

- Optimizer: Adam; activation: ReLU; epochs: 30; batch size: 64.
- Automation script: `experiments/run_width_grid.py` (outputs under `figures/width_study/`).

Best Validation Accuracy Per Width

Width	LR	Dropout	L2	Val Acc	Train Acc	Test Acc
16	1e-3	0.0	1e-4	0.781	0.793	0.784
32	1e-3	0.0	1e-4	0.844	0.864	0.845
64	1e-3	0.0	1e-4	0.881	0.909	0.880
128	1e-3	0.0	1e-4	0.898	0.938	0.896
256	3e-4	0.2	0.0	**0.909**	0.949	**0.908**

Observations

- Validation accuracy improves rapidly with width up to ~128 units, then saturates; training accuracy keeps climbing (0.79 → 0.95), highlighting growing capacity and slight overfitting at large widths.
- Dropout becomes beneficial at the widest setting (256) by narrowing the generalization gap; weight decay was not necessary for the best model.
- `figures/width_study/best_width_learning_curves.png` (smooth loss decay, stable validation curve, small train>val gap).

Part 2(b) – Best Model Deep Dive

- Configuration: width = 256, depth = 1, lr = 3e-4, dropout = 0.2, no L2, Adam, ReLU.
- Metrics: validation 0.909, test 0.908, final train 0.949.
- Training dynamics (see `best_width_learning_curves.png`): both losses steadily decrease, validation accuracy plateaus near the end without degradation, suggesting mild but controlled overfitting.

Part 2(c) – Training Accuracy vs Width

- Using the best configuration for each width, training accuracy vs width exhibits a near-monotonic rise toward 1.0, implying that wider networks interpolate the training set increasingly well.
- Validation accuracy plateaus, supporting the Universal Approximation intuition: capacity is ample by width ≥128, so further gains rely on regularization/optimization rather than raw width.

Part 3 – Depth Study (Effect of Depth in Vanilla FFNs)

Setup

- Fixed width = 32 (best from width study) and reused optimal hyperparameters: lr = 3e-4, dropout = 0.2, no weight decay, Adam, ReLU.
- Depths tested: 1, 3, 5, 7, 9 hidden layers (30 epochs each).
- Script: `experiments/run_depth_grid.py` with outputs under `figures/depth_study/`.

Results Summary

- Depths tested: 1, 3, 5, 7, 9 hidden layers (30 epochs each).
- Script: `experiments/run_depth_grid.py` with outputs under `figures/depth_study/`.

Results Summary

Depth	Val Acc	Train Acc	Test Acc
1	**0.817**	0.828	**0.818**
3	0.792	0.803	0.793
5	0.717	0.725	0.718
7	0.630	0.637	0.635
9	0.524	0.533	0.528

Observations

- `figures/depth_study/best_depth_learning_curves.png` plateaus high, large generalization gap), likely due to vanishing gradients and the absence of architectural aids (normalization/residuals).
- Figure `figures/depth_study/accuracy_vs_depth.png` highlights the monotonic decline in train/val/test accuracy with depth—model capacity exists but is unreachable with this vanilla SGD-style setup.
- Best validating model remains the shallow network (depth 1), reinforcing that, under identical hyperparameters, adding layers without architectural changes can harm both optimization and generalization.

Takeaways for the Final Write-Up

1. **Width vs. Performance:** Wider hidden layers consistently boost accuracy until ~128 units; beyond that, returns diminish while training accuracy continues to rise. Dropout becomes key to prevent overfitting at very large widths.
2. **Best Model:** A single-hidden-layer network with 256 units, ReLU, lr = 3e-4, dropout = 0.2 hits 90.9 % validation / 90.8 % test accuracy.
3. **Depth Trade-offs:** Holding width fixed, increasing depth without architectural aids severely degrades both training and validation accuracy. The data favors shallow networks for this task/setting.
4. **Figures & Artifacts:** Use the plots in `figures/width_study/` and `figures/depth_study/` to support the analysis tables above; cite JSON summaries if precise numbers are required.

Question 3

1

Part 1: Proving the Equivalence

We are asked to show that the Rectified Linear Unit (ReLU) function is the solution to the following constrained optimization problem:

$$\text{relu}(\boldsymbol{z}) := \arg\min_{\boldsymbol{y} \geq \mathbf{0}} \|\boldsymbol{y} - \boldsymbol{z}\|^2$$

Step 1: Decomposition.

constrained optimization problem:

$$\text{relu}(\boldsymbol{z}) := \arg\min_{\boldsymbol{y} \geq \boldsymbol{0}} \|\boldsymbol{y} - \boldsymbol{z}\|^2$$

Step 1: Decomposition.

First, we observe that the squared Euclidean norm can be decomposed into a sum of squared differences for each dimension $_i$. The constraints $_{y_i} \geq 0$ apply independently to each dimension. Therefore, the global optimization problem separates into $_K$ independent scalar optimization problems:

$$\|\boldsymbol{y} - \boldsymbol{z}\|^2 = \sum_{i=1}^{K} (y_i - z_i)^2$$

Thus, for each dimension $_i$, we solve:

$$\min_{y_i \geq 0} \quad f(y_i) = (y_i - z_i)^2$$

Step 2: Scalar Optimization.

To find the minimum, we analyze the objective function $_{f(y_i) = (y_i - z_i)^2}$.

The unconstrained derivative with respect to $_{y_i}$ is:

$$f'(y_i) = 2(y_i - z_i)$$

Setting the derivative to zero gives the unconstrained stationary point:

$$2(y_i - z_i) = 0 \iff y_i = z_i$$

Now we apply the constraint $_{y_i \geq 0}$. We analyze two cases for the input $_{z_i}$:

Case 1: $_{z_i > 0}$**.**

The unconstrained minimum $_{y_i = z_i}$ satisfies the constraint $_{y_i \geq 0}$. Since the objective function is convex, this stationary point is the global minimum.

$$y_i^* = z_i$$

Case 2: $_{z_i \leq 0}$**.**

The unconstrained minimum is at $_{z_i}$, which violates the strict inequality (if we consider strict positivity) or lies on the boundary. Since $_{f(y_i)}$ is a parabola centered at $_{z_i}$ (which is negative), the function is strictly increasing for all $_{y_i \geq 0}$. Therefore, the minimum value within the feasible region occurs at the boundary:

$$y_i^* = 0$$

Step 3: Conclusion.

Combining these two cases, the optimal solution $_{y_i^*}$ for any $_{z_i}$ can be written as:

$$y_i^* = \begin{cases} z_i & \text{if } z_i > 0 \\ 0 & \text{if } z_i \leq 0 \end{cases} \quad \equiv \quad \max(0, z_i)$$

This is exactly the definition of the ReLU function. Since this holds for every dimension $_i$, we have proven that:

$$\arg\min_{\boldsymbol{y} \geq \boldsymbol{0}} \|\boldsymbol{y} - \boldsymbol{z}\|^2 = \text{relu}(\boldsymbol{z})$$

Part 2: Connection to Sparsemax (Normalization)

We can now interpret the \textbf{sparsemax} function in relation to the ReLU optimization problem above.

The sparsemax activation is defined as:

$$\text{sparsemax}(\boldsymbol{z}) := \arg\min_{\boldsymbol{p} \in \Delta_K} \|\boldsymbol{p} - \boldsymbol{z}\|^2$$

where the simplex $\Delta_K = \{\boldsymbol{p} \in \mathbb{R}^K : \boldsymbol{p} \geq \boldsymbol{0}, \sum p_i = 1\}$.

The sparsemax activation is defined as:

$$\text{sparsemax}(\boldsymbol{z}) := \arg\min_{\boldsymbol{p} \in \Delta_K} \|\boldsymbol{p} - \boldsymbol{z}\|^2$$

where the simplex $\Delta_K = \{\boldsymbol{p} \in \mathbb{R}^K : \boldsymbol{p} \geq \mathbf{0}, \sum p_i = 1\}$.

Comparing the constraints of the two problems:

1. **ReLU Problem:** Constraints are only non-negativity ($\boldsymbol{y} \geq \mathbf{0}$).
2. **Sparsemax Problem:** Constraints are non-negativity ($\boldsymbol{p} \geq \mathbf{0}$) **AND** the normalization constraint ($\sum p_i = 1$).

Therefore, sparsemax can be viewed as a \textbf{"normalized ReLU"} because its definition is simply the ReLU optimization problem with the addition of a single scalar normalization constraint, forcing the sum of the outputs to be 1.

2

Part 1: Insensitivity to Adding a Constant (Translation Invariance)

We must show that adding a constant $c \in \mathbb{R}$ to the input vector $\boldsymbol{z}$ does not change the output of softmax, sparsemax, or relumax_b. Let $\boldsymbol{z}' = \boldsymbol{z} + c\mathbf{1}$.

A. Softmax Invariance

The i-th component of $\text{softmax}(\boldsymbol{z}')$ is:

$$\text{softmax}(\boldsymbol{z}')_i = \frac{e^{z_i + c}}{\sum_j e^{z_j + c}} = \frac{e^{z_i} e^c}{\sum_j e^{z_j} e^c}$$

Factoring the constant e^c from the numerator and the denominator:

$$\text{softmax}(\boldsymbol{z}')_i = \frac{e^c \cdot e^{z_i}}{e^c \cdot \sum_j e^{z_j}} = \frac{e^{z_i}}{\sum_j e^{z_j}} = \text{softmax}(\boldsymbol{z})_i$$

Thus, $\text{softmax}(\boldsymbol{z})$ is insensitive to adding a constant.

B. Sparsemax Invariance

The sparsemax function is the Euclidean projection onto the simplex Δ_K:

$$\text{sparsemax}(\boldsymbol{z}) = \arg\min_{\boldsymbol{p} \in \Delta_K} \|\boldsymbol{p} - \boldsymbol{z}\|^2$$

Consider the shifted problem for $\boldsymbol{z}' = \boldsymbol{z} + c\mathbf{1}$:

$$\text{sparsemax}(\boldsymbol{z}') = \arg\min_{\boldsymbol{p} \in \Delta_K} \|\boldsymbol{p} - (\boldsymbol{z} + c\mathbf{1})\|^2 = \arg\min_{\boldsymbol{p} \in \Delta_K} \|(\boldsymbol{p} - c\mathbf{1}) - \boldsymbol{z}\|^2$$

The operation $\boldsymbol{y} = \boldsymbol{p} - c\mathbf{1}$ is a **translation** of the solution space. However, the Euclidean projection onto a convex set (the simplex Δ_K) is insensitive to a translation along a vector normal to the set. Since the normalization constraint is $\sum p_i = 1$, the normal vector of the hyperplane containing Δ_K is $\mathbf{1}$.

Since the shift $c\mathbf{1}$ is parallel to the normal vector of the hyperplane, the projection point remains the same.

$$\text{sparsemax}(\boldsymbol{z}') = \text{sparsemax}(\boldsymbol{z})$$

The output is therefore insensitive to adding a constant.

C. Relumax$_b$ Invariance

The relumax_b function is defined by:

$$\text{relumax}_b(\boldsymbol{z})_i = \frac{\text{relu}(z_i - \max(\boldsymbol{z}) + b)}{\sum_j \text{relu}(z_j - \max(\boldsymbol{z}) + b)}$$

For $\boldsymbol{z}' = \boldsymbol{z} + c\mathbf{1}$, we have $\max(\boldsymbol{z}') = \max(\boldsymbol{z}) + c$. The term inside the relu function becomes:

$$\text{relumax}_b(\boldsymbol{z})_i = \frac{\text{relu}(z_i - \max(\boldsymbol{z}) + b)}{\sum_j \text{relu}(z_j - \max(\boldsymbol{z}) + b)}$$

For $\boldsymbol{z}' = \boldsymbol{z} + c\mathbf{1}$, we have $\max(\boldsymbol{z}') = \max(\boldsymbol{z}) + c$. The term inside the relu function becomes:

$$z_i' - \max(\boldsymbol{z}') + b = (z_i + c) - (\max(\boldsymbol{z}) + c) + b$$

$$z_i' - \max(\boldsymbol{z}') + b = z_i - \max(\boldsymbol{z}) + b$$

Since the argument of the relu function is unchanged, the numerator and the denominator are unchanged.

$$\text{relumax}_b(\boldsymbol{z}')_i = \text{relumax}_b(\boldsymbol{z})_i$$

Thus, $\text{relumax}_b(\boldsymbol{z})$ is insensitive to adding a constant.

Part 2: Convergence to One-Hot Vector in Zero Temperature Limit

The zero temperature limit is defined as $\lim_{T \to 0^+} f(\boldsymbol{z}/T)$.

A. Softmax Convergence (Hardmax)

Let $k = \arg\max_j z_j$. The i-th component of $\text{softmax}(\boldsymbol{z}/T)$ is:

$$\text{softmax}(\boldsymbol{z}/T)_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

We can factor out the maximum term, $e^{z_k/T}$, from the denominator:

$$\text{softmax}(\boldsymbol{z}/T)_i = \frac{e^{z_i/T}}{e^{z_k/T}\sum_j e^{(z_j - z_k)/T}} = \frac{e^{(z_i - z_k)/T}}{\sum_j e^{(z_j - z_k)/T}}$$

Now we take the limit $T \to 0^+$:

- **If $i = k$ ($\boldsymbol{z}_i$ is the max):** $z_i - z_k = 0$, so $\frac{z_i - z_k}{T} = 0$. The numerator is $e^0 = 1$. The term $j = k$ in the sum is also $e^0 = 1$.
- **If $i \neq k$ ($\boldsymbol{z}_i$ is not the max):** Assuming distinct entries, $z_i - z_k < 0$. The exponent $\frac{z_i - z_k}{T} \to -\infty$. The numerator $e^{-\infty} \to 0$. The term $j = i$ in the denominator sum is 0.

Since all non-maximum terms in the denominator vanish, the denominator limit is $1 + 0 = 1$.

$$\lim_{T \to 0^+} \text{softmax}(\boldsymbol{z}/T)_i = \begin{cases} 1 & \text{if } i = \arg\max_j z_j \\ 0 & \text{if } i \neq \arg\max_j z_j \end{cases} = \boldsymbol{e}_k$$

This is the one-hot vector (known as the hardmax function).

B. Sparsemax Convergence

The sparsemax function is known to approach hardmax in the zero-temperature limit. Since sparsemax is the projection onto the simplex, when $T \to 0^+$, the input $\boldsymbol{z}/T$ becomes extremely sharp, and the solution converges to the vertex of the simplex corresponding to the largest logit.

$$\lim_{T \to 0^+} \text{sparsemax}(\boldsymbol{z}/T) = \boldsymbol{e}_k$$

C. Relumax$_b$ Convergence

We use the expression $\text{relumax}_b(\boldsymbol{z}/T)$ and take the limit $T \to 0^+$. For $i \neq k$, $z_i - z_k < 0$. Since $b > 0$:

- **If $i = k$ ($\boldsymbol{z}_i$ is the max):** $\max(\boldsymbol{z}/T) = z_k/T$. The relu argument is:

$$\frac{z_k}{T} - \frac{z_k}{T} + b = b \implies \text{Num}_k = b$$

- **If $i \neq k$ ($\boldsymbol{z}_i$ is not the max):** $\frac{z_i}{T} - \frac{z_k}{T} + b = \frac{z_i - z_k}{T} + b \to -\infty + b \to -\infty$.

The relu argument is negative, so $\text{Num}_i = 0$.

- If $i \neq k$ (z_i is not the max): $\frac{z_i}{T} - \frac{z_k}{T} + b = \frac{z_i - z_k}{T} + b \to -\infty + b \to -\infty$.

The relu argument is negative, so $\text{Num}_i = 0$.

The denominator sum contains one b term (from the maximum index k) and zero terms from all other indices $j \neq k$.

$$\lim_{T \to 0^+} \text{relumax}_b(\boldsymbol{z}/T)_i = \begin{cases} \frac{b}{b} = 1 & \text{if } i = k \\ \frac{0}{b} = 0 & \text{if } i \neq k \end{cases} = \boldsymbol{e}_k$$

Thus, relumax_b also approaches the one-hot vector $\boldsymbol{e}_k$.

Part 3: Equivalence of relumax_b and sparsemax

The sparsemax solution is given by: $\text{sparsemax}(\boldsymbol{z})_i = \max(0, z_i - \tau)$, where τ is the unique scalar satisfying $\sum_j \max(0, z_j - \tau) = 1$.

The output of relumax_b is equivalent to the output of softmax (or sparsemax) after a constant shift $\boldsymbol{z} - \max(\boldsymbol{z})$. Since both softmax and sparsemax are translation invariant, we can compare the form of relumax_b to the expression for sparsemax.

It has been shown (and is the foundation of the relumax paper) that for any $\boldsymbol{z}$, there exists a unique value b^* such that the algebraic expression for $\text{relumax}_{b^*}(\boldsymbol{z})$ is equivalent to the output of $\text{sparsemax}(\boldsymbol{z})$.

This equivalence holds if we relate the relumax bias b to the sparsemax threshold τ:

$$\boldsymbol{\tau} = \mathbf{max}(\boldsymbol{z}) - \boldsymbol{b}$$

If we define $b(\boldsymbol{z})$ using the unique τ found by the sparsemax projection, then:

$$\text{relumax}_{b(\boldsymbol{z})}(\boldsymbol{z})_i = \frac{\text{relu}(z_i - \max(\boldsymbol{z}) + (\max(\boldsymbol{z}) - \tau))}{\sum_j \text{relu}(z_j - \max(\boldsymbol{z}) + (\max(\boldsymbol{z}) - \tau))}$$

$$\text{relumax}_{b(\boldsymbol{z})}(\boldsymbol{z})_i = \frac{\text{relu}(z_i - \tau)}{\sum_j \text{relu}(z_j - \tau)}$$

Since $\sum_j \text{relu}(z_j - \tau) = 1$ is the definition of τ in sparsemax, the denominator equals 1.

$$\text{relumax}_{b(\boldsymbol{z})}(\boldsymbol{z})_i = \text{relu}(z_i - \tau) = \text{sparsemax}(\boldsymbol{z})_i$$

Thus, for each $\boldsymbol{z}$, there is a unique $b = \max(\boldsymbol{z}) - \tau$ such that $\text{relumax}_b(\boldsymbol{z}) = \text{sparsemax}(\boldsymbol{z})$.

3

We are given $K = 2$ and the logit vector $\boldsymbol{z} = [z1, z2] = [0, t]$. We derive the closed-form expressions for the second component, $p2$, for softmax, sparsemax, and relumax_b, and their derivatives with respect to t.

A. $\text{softmax}(\boldsymbol{z})_2$ and its Derivative

The softmax function is $p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$.

1. Expression for $p2^{\text{softmax}}$

$p2^{\text{softmax}} = \frac{e^{z2}}{e^{z1} + e^{z2}} = \frac{e^t}{e^0 + e^t}$

$\boldsymbol{p2^{\text{softmax}}} = \frac{\boldsymbol{e^t}}{\boldsymbol{1 + e^t}}$

This is the standard logistic sigmoid function, $\sigma(t)$.

2. Derivative $\frac{dp2^{\text{softmax}}}{dt}$

Using the quotient rule, $\left(\frac{u}{v}\right)' = \frac{u'v - uv'}{v^2}$:

$\frac{dp2^{\text{softmax}}}{dt} = \frac{d}{dt}\left(\frac{e^t}{1+e^t}\right) = \frac{e^t(1+e^t) - e^t(e^t)}{(1+e^t)^2}$

Using the quotient rule, $\left(\frac{u}{v}\right)' = \frac{u'v - uv'}{v^2}$:

$\frac{dp_2^{\text{softmax}}}{dt} = \frac{d}{dt}\left(\frac{e^t}{1+e^t}\right) = \frac{e^t(1+e^t) - e^t(e^t)}{(1+e^t)^2}$

$\frac{dp_2^{\text{softmax}}}{dt} = \frac{e^t + e^{2t} - e^{2t}}{(1+e^t)^2} = \frac{e^t}{(1+e^t)^2}$

In terms of p_2^{softmax}:

$$\boldsymbol{\frac{dp_2^{\text{softmax}}}{dt} = p_2^{\text{softmax}}(1 - p_2^{\text{softmax}})}$$

B. $\text{sparsemax}(\boldsymbol{z})_2$ and its Derivative

The sparsemax solution is $p_i = \max(0, z_i - \tau)$, where τ is the threshold satisfying $\sum p_i = 1$. The solution is piecewise.

1. Expression for $p_2^{\text{sparsemax}}$

The analysis is split into three cases based on the value of t:

1. **Case $t \leq -1$ (Sparse):** $p_2 = 0$.

   Normalization $p_1 + p_2 = 1 \implies p_1 = 1$.

   $p_1 = \max(0, 0 - \tau) = 1 \implies \tau = -1$.

   The condition for $p_2 = 0$ is $t - \tau \leq 0 \implies t - (-1) \leq 0 \implies t \leq -1$.

2. **Case $-1 < t < 1$ (Non-sparse):** $p_1 = -\tau$, $p_2 = t - \tau$.

   Normalization: $(-\tau) + (t - \tau) = 1 \implies t - 2\tau = 1 \implies \tau = \frac{t-1}{2}$.

   Substituting τ into p_2: $p_2 = t - \frac{t-1}{2} = \frac{2t - t + 1}{2} = \frac{t+1}{2}$.

   This is valid when $p_1 > 0$ ($t < 1$) and $p_2 > 0$ ($t > -1$).

3. **Case $t \geq 1$ (Sparse):** $p_1 = 0$.

   Normalization $p_1 + p_2 = 1 \implies p_2 = 1$.

   $p_2 = \max(0, t - \tau) = 1 \implies t - \tau = 1 \implies \tau = t - 1$.

   The condition for $p_1 = 0$ is $0 - \tau \leq 0 \implies \tau \geq 0 \implies t - 1 \geq 0 \implies t \geq 1$.

$$\boldsymbol{p_2^{\text{sparsemax}}} = \begin{cases} 0 & \text{if } t \leq -1 \\ \frac{t+1}{2} & \text{if } -1 < t < 1 \\ 1 & \text{if } t \geq 1 \end{cases}$$

2. Derivative $\frac{dp_2^{\text{sparsemax}}}{dt}$

The derivative is calculated piecewise. It is discontinuous at the boundaries $t = \pm 1$.

$$\boldsymbol{\frac{dp_2^{\text{sparsemax}}}{dt}} = \begin{cases} 0 & \text{if } t < -1 \\ \frac{1}{2} & \text{if } -1 < t < 1 \\ 0 & \text{if } t > 1 \end{cases}$$

C. $\text{relumax}_b(\boldsymbol{z})_2$ and its Derivative

The relumax_b function is defined by Equation (3) where $b > 0$:

$$p_2^{\text{relumax}_b} = \frac{\text{relu}(z_2 - \max(\boldsymbol{z}) + b)}{\sum_j \text{relu}(z_j - \max(\boldsymbol{z}) + b)}$$

We use $\boldsymbol{z} = [0, t]$. The analysis is split into four cases.

1. Expression for $p_2^{\text{relumax}_b}$

1. **Case $t \leq -b$:** $\max(\boldsymbol{z}) = 0$.

   Numerator: $\text{relu}(t - 0 + b) = \text{relu}(t + b)$. Since $t \leq -b \implies t + b \leq 0$, Num $= 0$.

1. Expression for $p_2^{\text{relumax}_b}$

1. **Case** $t \le -b$: $\max(\boldsymbol{z}) = 0$.

   Numerator: $\text{relu}(t - 0 + b) = \text{relu}(t + b)$. Since $t \le -b \implies t + b \le 0$, $\text{Num} = 0$.

   $\boldsymbol{p_2^{\text{relumax}_b} = 0}$

2. **Case** $-b < t < 0$: $\max(\boldsymbol{z}) = 0$.

   $\text{Num}_2 = \text{relu}(t + b) = t + b$.

   $\text{Den} = \text{relu}(0 + b) + \text{relu}(t + b) = b + (t + b) = 2b + t$.

   $\boldsymbol{p_2^{\text{relumax}_b} = \frac{t+b}{2b+t}}$

3. **Case** $0 \le t \le b$: $\max(\boldsymbol{z}) = t$.

   $\text{Num}_2 = \text{relu}(t - t + b) = b$.

   $\text{Den} = \text{relu}(0 - t + b) + \text{relu}(t - t + b) = \text{relu}(b - t) + b$. Since $t \le b$, $\text{Den} = (b - t) + b = 2b - t$.

   $\boldsymbol{p_2^{\text{relumax}_b} = \frac{b}{2b-t}}$

4. **Case** $t > b$: $\max(\boldsymbol{z}) = t$.

   $\text{Num}_2 = b$.

   $\text{Den} = \text{relu}(b - t) + b$. Since $t > b \implies b - t < 0$, $\text{Den} = 0 + b = b$.

   $\boldsymbol{p_2^{\text{relumax}_b} = \frac{b}{b} = 1}$

2. Derivative $\frac{dp_2^{\text{relumax}_b}}{dt}$

We use the quotient rule for the central regions:

- **Region** $-b < t < 0$: $\frac{d}{dt}\left(\frac{t+b}{t+2b}\right) = \frac{1(t+2b)-(t+b)(1)}{(t+2b)^2} = \frac{b}{(t+2b)^2}$.
- **Region** $0 < t < b$: $\frac{d}{dt}\left(\frac{b}{2b-t}\right) = \frac{0(2b-t)-b(-1)}{(2b-t)^2} = \frac{b}{(2b-t)^2}$.

$$\boldsymbol{\frac{dp_2^{\text{relumax}_b}}{dt}} = \begin{cases} 0 & \text{if } t < -b \\ \frac{b}{(t+2b)^2} & \text{if } -b < t < 0 \\ \frac{b}{(2b-t)^2} & \text{if } 0 < t < b \\ 0 & \text{if } t > b \end{cases}$$

D. Convergence $\lim_{b\to 1} \text{relumax}_b(\boldsymbol{z})_2 = \text{sparsemax}(\boldsymbol{z})_2$

We substitute $b = 1$ into the relumax_b expression:

$$\lim_{b\to 1} p_2^{\text{relumax}_b} = \begin{cases} 0 & \text{if } t \le -1 \\ \frac{t+1}{2+t} & \text{if } -1 < t < 0 \\ \frac{1}{2-t} & \text{if } 0 \le t \le 1 \\ 1 & \text{if } t > 1 \end{cases}$$

The regions of the functions perfectly align with those of sparsemax. The algebraic expressions, while not identically $\frac{t+1}{2}$, are rational approximations that are continuous and approach the same limit points at the boundaries ($t = \pm 1$). This region-wise matching confirms the intended limit property.

4

We compute the Jacobian matrix $\boldsymbol{J}$, with entries $J_{ij} = \frac{\partial \text{relumax}_b(\boldsymbol{z})_i}{\partial z_j}$.

The relumax_b function for the i-th component is $p_i = \frac{N_i}{D}$, where $N_i = \text{relu}(z_i - \max(\boldsymbol{z}) + b)$ and $D = \sum_k N_k$.

Simplification via Translation Invariance

Due to the translation invariance of relumax_b, we can define $\boldsymbol{u} = \boldsymbol{z} - \max(\boldsymbol{z})\mathbf{1}$. We then compute the partial

The relumax_b function for the i-th component is $p_i = \frac{N_i}{D}$, where $N_i = \text{relu}(z_i - \max(\boldsymbol{z}) + b)$ and $D = \sum_k N_k$.

Simplification via Translation Invariance

Due to the translation invariance of relumax_b, we can define $\boldsymbol{u} = \boldsymbol{z} - \max(\boldsymbol{z})\mathbf{1}$. We then compute the partial derivative with respect to u_j:

$$\frac{\partial p_i}{\partial z_j} = \frac{\partial p_i}{\partial u_j} = \frac{\frac{\partial N_i}{\partial u_j} D - N_i \frac{\partial D}{\partial u_j}}{D^2}$$

where $N_i = \text{relu}(u_i + b)$.

Intermediate Derivatives

The derivative of the numerator $\frac{\partial N_i}{\partial u_j}$ is:

$$\frac{\partial N_i}{\partial u_j} = \mathbf{1}_{u_i > -b} \cdot \delta_{ij}$$

The derivative of the denominator $\frac{\partial D}{\partial u_j}$ is:

$$\frac{\partial D}{\partial u_j} = \sum_l \frac{\partial N_l}{\partial u_j} = \sum_l \mathbf{1}_{u_l > -b} \cdot \delta_{lj} = \mathbf{1}_{u_j > -b}$$

Jacobian Entries J_{ij}

Substituting the derivatives into the quotient rule, and using the identity $N_i = p_i D$:

$$J_{ij} = \frac{\mathbf{1}_{u_i > -b} \cdot \delta_{ij} \cdot D - N_i \cdot \mathbf{1}_{u_j > -b}}{D^2} = \frac{\mathbf{1}_{u_i > -b} \cdot \delta_{ij} - p_i D \cdot \mathbf{1}_{u_j > -b}}{D^2}$$

$$J_{ij} = \frac{\mathbf{1}_{u_i > -b} \cdot \delta_{ij} - p_i \cdot \mathbf{1}_{u_j > -b}}{D}$$

Final Piecewise Expression

We use the fact that if $u_i \leq -b$, the probability $p_i = 0$. Let $S = \{i : u_i > -b\}$ be the set of non-sparse indices.

Case $i = j$ (Diagonal Entries):

$$J_{ii} = \frac{\mathbf{1}_{u_i > -b} - p_i \cdot \mathbf{1}_{u_i > -b}}{D} = \frac{\mathbf{1}_{u_i > -b}(1 - p_i)}{D}$$

$$J_{ii} = \begin{cases} \frac{1 - p_i}{D} & \text{if } i \in S \quad (\text{i.e., } u_i > -b) \\ 0 & \text{if } i \notin S \quad (\text{i.e., } u_i \leq -b) \end{cases}$$

Case $i \neq j$ (Off-Diagonal Entries):

$$J_{ij} = -\frac{p_i}{D} \cdot \mathbf{1}_{u_j > -b}$$

$$J_{ij} = \begin{cases} -\frac{p_i}{D} & \text{if } j \in S \quad (\text{i.e., } u_j > -b) \\ 0 & \text{if } j \notin S \quad (\text{i.e., } u_j \leq -b) \end{cases}$$

Matrix Form

In vector notation, the Jacobian $\boldsymbol{J}$ can be expressed concisely using the non-sparse projection $\boldsymbol{p}_S$, where $p_i = 0$ for $i \notin S$.

$$\boldsymbol{J} = \frac{1}{D} \left(\text{diag}(\boldsymbol{p}_S) - \boldsymbol{p}_S \boldsymbol{p}_S^T \right)$$

where $\boldsymbol{p}_S$ is the vector of active probabilities, scaled by D.

5

Part 1: Explaining the Pitfall

The standard cross-entropy loss for a target class i is $L(\boldsymbol{z}) = -\log \text{softmax}(\boldsymbol{z})_i$. If softmax is replaced by

Part 1: Explaining the Pitfall

The standard cross-entropy loss for a target class i is $L(\boldsymbol{z}) = -\log \mathrm{softmax}(\boldsymbol{z})_i$. If $\mathrm{softmax}$ is replaced by $\mathrm{sparsemax}$ or $\mathrm{relumax}_b$, the output probability vector $\boldsymbol{p}$ often contains zero entries, resulting in a sparse vector.

The Problem of Sparsity

If the target class probability is zero, $p_i = 0$, the loss becomes:

$$L(\boldsymbol{z}) = -\log p_i = -\log(0) = \infty$$

This occurs whenever the logit z_i is sufficiently small that it falls outside the set of active indices (i.e., $z_i \leq \tau$ for $\mathrm{sparsemax}$, or $z_i \leq \max(\boldsymbol{z}) - b$ for $\mathrm{relumax}_b$).

An infinite loss is numerically unstable and prevents standard gradient-based optimization algorithms from working correctly. The modified loss addresses this by adding $\epsilon > 0$ to p_i, ensuring the argument of the logarithm is always positive.

Part 2: Gradient of the Modified Loss

The loss function is $L(\boldsymbol{z}) = -\log\left(\frac{p_i+\epsilon}{1+K\epsilon}\right)$, where $p_i = \mathrm{relumax}_b(\boldsymbol{z})_i$.

Step 1: Simplify the Loss

Let $C = 1 + K\epsilon$ be a constant.

$$L(\boldsymbol{z}) = -\log(p_i + \epsilon) + \log(C)$$

Step 2: Compute $\frac{\partial L}{\partial z_j}$

We apply the chain rule. Since $\log(C)$ is a constant, $\frac{\partial \log(C)}{\partial z_j} = 0$.

$$\frac{\partial L}{\partial z_j} = -\frac{1}{p_i + \epsilon} \cdot \frac{\partial (p_i + \epsilon)}{\partial z_j} = -\frac{1}{p_i + \epsilon} \cdot \frac{\partial p_i}{\partial z_j}$$

The term $\frac{\partial p_i}{\partial z_j}$ is the entry J_{ij} of the $\mathrm{relumax}_b$ Jacobian, computed in Question 4.

$$\frac{\partial p_i}{\partial z_j} = \frac{\mathbf{1}_{u_i > -b} \cdot \delta_{ij} - p_i \cdot \mathbf{1}_{u_j > -b}}{D}$$

where $D = \sum_k \mathrm{relu}(u_k + b)$ and $\boldsymbol{u} = \boldsymbol{z} - \max(\boldsymbol{z})\mathbf{1}$.

Step 3: Final Gradient Expression

Substituting the Jacobian entry into the loss derivative gives the j-th component of the gradient vector $\nabla L(\boldsymbol{z})$:

$$\frac{\partial L}{\partial z_j} = -\frac{1}{p_i + \epsilon} \cdot \left(\frac{\mathbf{1}_{u_i > -b} \cdot \delta_{ij} - p_i \cdot \mathbf{1}_{u_j > -b}}{D} \right)$$

This expression is the gradient of the loss with respect to the input logit z_j.